



アプリケーション実行フックを管理します

Astra Control Center

NetApp
November 21, 2023

目次

アプリケーション実行フックを管理します	1
実行フックのタイプ	1
カスタム実行フックに関する重要な注意事項	1
既存の実行フックを表示します	3
既存のスクリプトを表示します	4
スクリプトを追加します	4
スクリプトを削除します	5
カスタム実行フックを作成します	5
実行フックの状態を確認します	6
スクリプトの使用状況を表示します	6
実行フックを無効にします	7
実行フックを削除します	7
実行フックの例	7

アプリケーション実行フックを管理します

実行フックは、管理対象アプリケーションのデータ保護操作と組み合わせて実行するように構成できるカスタムアクションです。たとえば、データベースアプリケーションがある場合、実行フックを使用して、スナップショットの前にすべてのデータベーストランザクションを一時停止し、スナップショットの完了後にトランザクションを再開できます。これにより、アプリケーションと整合性のある Snapshot を作成できます。

実行フックのタイプ

Astra Controlは、実行可能なタイミングに基づいて、次の種類の実行フックをサポートします。

- Snapshot前
- Snapshot後
- バックアップ前
- バックアップ後
- リストア後のPOSTコマンドです

カスタム実行フックに関する重要な注意事項

アプリケーションの実行フックを計画するときは、次の点を考慮してください。

- 実行フックは、スクリプトを使用してアクションを実行する必要があります。多くの実行フックは、同じスクリプトを参照できます。
- Astra Controlでは、実行フックが実行可能なシェルスクリプトの形式で記述されるようにするスクリプトが必要です。
- スクリプトのサイズは96KBに制限されています。
- Astra Controlは、実行フックの設定と一致条件を使用して、スナップショット、バックアップ、または復元操作に適用できるフックを決定します。
- 実行フックの障害はすべて'ソフトな障害'ですフックが失敗しても'他のフックとデータ保護操作は試行されます'ただし、フックが失敗すると、* アクティビティ * ページイベントログに警告イベントが記録されます。
- 実行フックを作成、編集、または削除するには、Owner、Admin、または Member 権限を持つユーザーである必要があります。
- 実行フックの実行に 25 分以上かかる場合 'フックは失敗し' 戻りコードが N/A のイベント・ログ・エントリが作成されます該当する Snapshot はタイムアウトして失敗とマークされ、タイムアウトを通知するイベントログエントリが生成されます。
- アドホックデータ保護操作の場合、すべてのフックイベントが生成され、[Activity]ページのイベントログに保存されます。ただし、スケジュールされたデータ保護処理については、フック障害イベントだけがイベントログに記録されます（スケジュールされたデータ保護処理自体によって生成されたイベントは記録されたままです）。



実行フックは、実行中のアプリケーションの機能を低下させるか、完全に無効にすることが多いため、カスタム実行フックの実行時間を最小限に抑えるようにしてください。実行フックが関連付けられている状態でバックアップまたはスナップショット操作を開始した後、キャンセルした場合でもバックアップまたはスナップショット操作がすでに開始されていればフックは実行できますつまり、バックアップ後の実行フックでは、バックアップが完了したとは判断できません。

実行順序

データ保護操作を実行すると、実行フックイベントが次の順序で実行されます。

1. 適用可能なカスタムプリオペレーション実行フックは、適切なコンテナで実行されます。カスタムのプリオペレーションフックは必要なだけ作成して実行できますが、操作前のこれらのフックの実行順序は保証も構成もされていません。
2. データ保護処理が実行されます。
3. 適用可能なカスタムポストオペレーション実行フックは、適切なコンテナで実行されます。必要な数のカスタムポストオペレーションフックを作成して実行できますが、操作後のこれらのフックの実行順序は保証されず、設定もできません。

同じ種類の実行フック（スナップショット前など）を複数作成する場合、これらのフックの実行順序は保証されません。ただし、異なるタイプのフックの実行順序は保証されています。たとえば、5つの異なるタイプのフックをすべて持つ構成の実行順序は、次のようになります。

1. 予備フックが実行されます
2. スナップショット前フックが実行されます
3. スナップショット後フックが実行されます
4. バックアップ後のフックが実行されます
5. 復元後のフックが実行されます

シナリオ番号2のこの設定の例は、の表を参照してください [\[フックが実行されるかどうかを確認します\]](#)。



本番環境で実行スクリプトを有効にする前に、必ず実行フックスクリプトをテストしてください。'kubectl exec' コマンドを使用すると、スクリプトを簡単にテストできます。本番環境で実行フックを有効にした後、作成されたSnapshotとバックアップをテストして整合性があることを確認します。これを行うには、アプリケーションを一時的な名前スペースにクローニングし、スナップショットまたはバックアップをリストアしてから、アプリケーションをテストします。

フックが実行されるかどうかを確認します

次の表を使用して、アプリケーションでカスタム実行フックが実行されるかどうかを判断します。

アプリケーションの高レベルの処理は、すべてスナップショット、バックアップ、またはリストアの基本的な処理のいずれかを実行することで構成されることに注意してください。シナリオによっては、クローニング処理はこれらの処理のさまざまな組み合わせで構成されるため、クローン処理を実行する実行フックはさまざまです。

In Place リストア処理では既存のSnapshotまたはバックアップが必要になるため、これらの処理ではSnapshot

またはバックアップフックは実行されません。

開始してスナップショットを含むバックアップをキャンセルし、実行フックが関連付けられている場合は、一部のフックが実行され、ほかのフックが実行されないことがあります。つまり、バックアップ後の実行フックでは、バックアップが完了したとは判断できません。キャンセルしたバックアップに関連する実行フックがある場合は、次の点に注意してください。



- バックアップ前およびバックアップ後のフックは常に実行されます。
- バックアップに新しいスナップショットが含まれており、スナップショットが開始されている場合は、スナップショット前フックとスナップショット後フックが実行されます。
- スナップショットの開始前にバックアップがキャンセルされた場合は、スナップショット前フックとスナップショット後フックは実行されません。

シナリオ (Scenario)	操作	既存のSnapshot	既存のバックアップ	ネームスペース	クラスター	スナップショットフックが実行されます	バックアップフックが実行されます	フックを元に戻します
1.	クローン	N	N	新規	同じ	Y	N	Y
2.	クローン	N	N	新規	違う	Y	Y	Y
3.	クローン またはリストア	Y	N	新規	同じ	N	N	Y
4.	クローン またはリストア	N	Y	新規	同じ	N	N	Y
5.	クローン またはリストア	Y	N	新規	違う	N	Y	Y
6.	クローン またはリストア	N	Y	新規	違う	N	N	Y
7.	リストア	Y	N	既存	同じ	N	N	Y
8.	リストア	N	Y	既存	同じ	N	N	Y
9.	スナップショット	該当なし	該当なし	該当なし	該当なし	Y	該当なし	該当なし
10.	バックアップ	N	該当なし	該当なし	該当なし	Y	Y	該当なし
11.	バックアップ	Y	該当なし	該当なし	該当なし	N	Y	該当なし

既存の実行フックを表示します

アプリケーションの既存のカスタム実行フックを表示できます。

手順

1. 「* アプリケーション」に移動し、管理アプリの名前を選択します。
2. [実行フック*] タブを選択します。

有効または無効になっているすべての実行フックを結果リストに表示できます。フックのステータス、ソース、および実行時刻（プリ/ポストオペレーション）を表示できます。実行フックに関連するイベントログを表示するには、左側のナビゲーション領域の * アクティビティ * ページに移動します。

既存のスクリプトを表示します

アップロードされた既存のスクリプトを表示できます。このページでは、使用中のスクリプトと、使用中のフックを確認することもできます。

手順

1. 「アカウント」に移動します。
2. [スクリプト]タブを選択します。

このページには、アップロードされた既存のスクリプトのリストが表示されます。[使用者*]列には、各スクリプトを使用している実行フックが表示されます。

スクリプトを追加します

実行フックが参照できるスクリプトを1つ以上追加できます。多くの実行フックは、同じスクリプトを参照できます。これにより、1つのスクリプトのみを変更することで、多数の実行フックを更新できます。

手順

1. 「アカウント」に移動します。
2. [スクリプト]タブを選択します。
3. 「* 追加」を選択します。
4. 次のいずれかを実行します。
 - カスタムスクリプトをアップロードする。
 - i. [ファイルのアップロード（Upload file）] オプションを選択します。
 - ii. ファイルを参照してアップロードします。
 - iii. スクリプトに一意の名前を付けます。
 - iv. （オプション）他の管理者がスクリプトについて知っておく必要があるメモを入力します。
 - v. 「スクリプトを保存」を選択します。
 - クリップボードからカスタムスクリプトを貼り付けます。
 - i. [貼り付け（Paste）]または[タイプ（* type）]オプションを選択する
 - ii. テキストフィールドを選択し、スクリプトテキストをフィールドに貼り付けます。
 - iii. スクリプトに一意の名前を付けます。
 - iv. （オプション）他の管理者がスクリプトについて知っておく必要があるメモを入力します。

5. 「スクリプトを保存」を選択します。

結果

新しいスクリプトが、[スクリプト]タブのリストに表示されます。

スクリプトを削除します

不要になって実行フックで使用されなくなったスクリプトは、システムから削除できます。

手順

1. 「アカウント」に移動します。
2. [スクリプト]タブを選択します。
3. 削除するスクリプトを選択し、「アクション」列のメニューを選択します。
4. 「* 削除」を選択します。



スクリプトが1つまたは複数の実行フックに関連付けられている場合、*Delete*アクションは使用できません。スクリプトを削除するには、まず関連する実行フックを編集し、別のスクリプトに関連付けます。

カスタム実行フックを作成します

アプリケーションのカスタム実行フックを作成できます。を参照してください ["実行フックの例"](#) フックの例を参照してください。実行フックを作成するには、Owner、Admin、または Member のいずれかの権限が必要です。



実行フックとして使用するカスタムシェルスクリプトを作成する場合は、特定のコマンドを実行するか、実行可能ファイルへの完全パスを指定する場合を除き、ファイルの先頭に適切なシェルを指定するようにしてください。

手順

1. 「* アプリケーション」を選択し、管理アプリの名前を選択します。
2. [実行フック*]タブを選択します。
3. 「* 追加」を選択します。
4. フックの詳細* (* Hook Details) 領域で、*操作 (* Operation *) ドロップダウンメニューから操作タイプを選択して、フックを実行するタイミングを決定します。
5. フックの一意の名前を入力します。
6. (オプション) 実行中にフックに渡す引数を入力し、各引数を入力した後で Enter キーを押して、それぞれを記録します。
7. [* Container Images * (コンテナイメージ*)] 領域で、アプリケーションに含まれるすべてのコンテナイメージに対してフックを実行する必要がある場合は、[* Apply to all container images * (すべてのコンテナイメージに適用*)] チェックボックスを有効にします。代わりに、フックが 1 つ以上の指定されたコンテナイメージに対してのみ機能する場合は、* Container image names to match * フィールドにコンテナイメージ名を入力します。
8. [* スクリプト * (* Script *)] 領域で、次のいずれかを実行します。

- 新しいスクリプトを追加します。
 - i. 「* 追加」を選択します。
 - ii. 次のいずれかを実行します。
 - カスタムスクリプトをアップロードする。
 - I. [ファイルのアップロード（Upload file）] オプションを選択します。
 - II. ファイルを参照してアップロードします。
 - III. スクリプトに一意の名前を付けます。
 - IV. （オプション）他の管理者がスクリプトについて知っておく必要があるメモを入力します。
 - V. 「スクリプトを保存」を選択します。
 - クリップボードからカスタムスクリプトを貼り付けます。
 - I. [貼り付け（Paste）]または[タイプ（* type）]オプションを選択する
 - II. テキストフィールドを選択し、スクリプトテキストをフィールドに貼り付けます。
 - III. スクリプトに一意の名前を付けます。
 - IV. （オプション）他の管理者がスクリプトについて知っておく必要があるメモを入力します。
- リストから既存のスクリプトを選択します。

このスクリプトを使用するように実行フックに指示します。

9. [* フックを追加*]を選択します。

実行フックの状態を確認します

スナップショット、バックアップ、または復元操作の実行が終了したら、操作の一部として実行された実行フックの状態を確認できます。このステータス情報を使用して、実行フックを保持するか、変更するか、削除するかを決定できます。

手順

1. 「* アプリケーション」を選択し、管理アプリの名前を選択します。
2. [データ保護]タブを選択します。
3. 実行中のSnapshotを表示するには「* Snapshots」を選択し、実行中のバックアップを表示するには「* Backups」を選択します。

フック状態*は、操作完了後の実行フックランのステータスを示します。状態にカーソルを合わせると、詳細を確認できます。たとえば、スナップショット中に実行フック障害が発生した場合、そのスナップショットのフック状態にカーソルを合わせると、失敗した実行フックのリストが表示されます。各失敗の理由を確認するには、左側のナビゲーション領域の*アクティビティ*ページを確認します。

スクリプトの使用状況を表示します

どの実行フックがAstra Control Web UIの特定のスクリプトを使用しているかを確認できます。

手順

1. 「* アカウント *」を選択します。
2. [スクリプト]タブを選択します。

スクリプトのリストにある* Used by *列には、リスト内の各スクリプトを使用しているフックの詳細が表示されます。

3. 目的のスクリプトの[使用者*]列の情報を選択します。

より詳細なリストが表示され、スクリプトを使用しているフックの名前と、それらが実行されるように構成されている操作のタイプが示されます。

実行フックを無効にします

アプリケーションのスナップショットの前または後に実行を一時的に禁止する場合は、実行フックを無効にできます。実行フックを無効にするには、Owner、Admin、またはMemberのいずれかの権限が必要です。

手順

1. 「* アプリケーション」を選択し、管理アプリの名前を選択します。
2. [実行フック*]タブを選択します。
3. 無効にするフックの* アクション*列のオプションメニューを選択します。
4. [Disable]を選択します。

実行フックを削除します

不要になった実行フックは完全に削除できます。実行フックを削除するには、Owner、Admin、またはMemberのいずれかの権限が必要です。

手順

1. 「* アプリケーション」を選択し、管理アプリの名前を選択します。
2. [実行フック*]タブを選択します。
3. 削除するフックの* アクション*列のオプションメニューを選択します。
4. 「* 削除」を選択します。

実行フックの例

次の例を使用して、実行フックの構造を確認してください。これらのフックは、テンプレートまたはテストスクリプトとして使用できます。

シンプルな成功例

次に、成功し、標準出力および標準エラーにメッセージを書き込む単純フックの例を示します。

```
#!/bin/sh

# success_sample.sh
#
# A simple noop success hook script for testing purposes.
#
# args: None
#

#
# Writes the given message to standard output
#
# $* - The message to write
#
msg() {
    echo "$*"
}

#
# Writes the given information message to standard output
#
# $* - The message to write
#
info() {
    msg "INFO: $*"
}

#
# Writes the given error message to standard error
#
# $* - The message to write
#
error() {
    msg "ERROR: $*" 1>&2
}

#
# main
#

# log something to stdout
info "running success_sample.sh"

# exit with 0 to indicate success
```

```
info "exit 0"  
exit 0
```

シンプルな成功の例（**bash** バージョン）

次に、bash 用にかかれた標準出力と標準エラーにメッセージを書き込む単純なフックの例を示します。

```
#!/bin/bash  
  
# success_sample.bash  
#  
# A simple noop success hook script for testing purposes.  
#  
# args: None  
  
#  
# Writes the given message to standard output  
#  
# $* - The message to write  
#  
msg() {  
    echo "$*"
}  
  
#  
# Writes the given information message to standard output  
#  
# $* - The message to write  
#  
info() {  
    msg "INFO: $*"
}  
  
#  
# Writes the given error message to standard error  
#  
# $* - The message to write  
#  
error() {  
    msg "ERROR: $*" 1>&2
}
```

```
#
# main
#

# log something to stdout
info "running success_sample.bash"

# exit with 0 to indicate success
info "exit 0"
exit 0
```

単純な成功例（zsh バージョン）

これは、成功した単純なフックの例であり、標準出力と標準エラーに Z シェル用に記述されたメッセージを書き込みます。

```
#!/bin/zsh

# success_sample.zsh
#
# A simple noop success hook script for testing purposes.
#
# args: None
#

#
# Writes the given message to standard output
#
# $* - The message to write
#
msg() {
    echo "$*"
}

#
# Writes the given information message to standard output
#
# $* - The message to write
#
info() {
    msg "INFO: $*"
}

#
```

```

# Writes the given error message to standard error
#
# $* - The message to write
#
error() {
    msg "ERROR: $*" 1>&2
}

#
# main
#

# log something to stdout
info "running success_sample.zsh"

# exit with 0 to indicate success
info "exit 0"
exit 0

```

引数を指定した成功の例

次の例は、フックで args を使用する方法を示しています。

```

#!/bin/sh

# success_sample_args.sh
#
# A simple success hook script with args for testing purposes.
#
# args: Up to two optional args that are echoed to stdout
#
# Writes the given message to standard output
#
# $* - The message to write
#
msg() {
    echo "$*"
}

#
# Writes the given information message to standard output
#
# $* - The message to write

```

```

#
info() {
    msg "INFO: $*"
}

#
# Writes the given error message to standard error
#
# $* - The message to write
#
error() {
    msg "ERROR: $*" 1>&2
}

#
# main
#

# log something to stdout
info "running success_sample_args.sh"

# collect args
arg1=$1
arg2=$2

# output args and arg count to stdout
info "number of args: $#\"
info "arg1 ${arg1}\"
info "arg2 ${arg2}\"

# exit with 0 to indicate success
info "exit 0\"
exit 0

```

スナップショット前 / スナップショット後のフックの例

次の例は、Snapshot 前フックと Snapshot 後フックの両方に同じスクリプトを使用する方法を示しています。

```

#!/bin/sh

# success_sample_pre_post.sh
#
# A simple success hook script example with an arg for testing purposes

```

```

# to demonstrate how the same script can be used for both a prehook and
posthook
#
# args: [pre|post]

# unique error codes for every error case
ebase=100
eusage=$((ebase+1))
ebadstage=$((ebase+2))
epre=$((ebase+3))
epost=$((ebase+4))

#
# Writes the given message to standard output
#
# $* - The message to write
#
msg() {
    echo "$*"
}

#
# Writes the given information message to standard output
#
# $* - The message to write
#
info() {
    msg "INFO: $*"
}

#
# Writes the given error message to standard error
#
# $* - The message to write
#
error() {
    msg "ERROR: $*" 1>&2
}

#
# Would run prehook steps here
#
prehook() {

```

```

    info "Running noop prehook"
    return 0
}

#
# Would run posthook steps here
#
posthook() {
    info "Running noop posthook"
    return 0
}

#
# main
#

# check arg
stage=$1
if [ -z "${stage}" ]; then
    echo "Usage: $0 <pre|post>"
    exit ${eusage}
fi

if [ "${stage}" != "pre" ] && [ "${stage}" != "post" ]; then
    echo "Invalid arg: ${stage}"
    exit ${ebadstage}
fi

# log something to stdout
info "running success_sample_pre_post.sh"

if [ "${stage}" = "pre" ]; then
    prehook
    rc=$?
    if [ ${rc} -ne 0 ]; then
        error "Error during prehook"
    fi
fi

if [ "${stage}" = "post" ]; then
    posthook
    rc=$?
    if [ ${rc} -ne 0 ]; then
        error "Error during posthook"
    fi
fi

```



```
fi

exit ${rc}
```

失敗の例

次の例は、フックで障害を処理する方法を示しています。

```
#!/bin/sh

# failure_sample_arg_exit_code.sh
#
# A simple failure hook script for testing purposes.
#
# args: [the exit code to return]
#

#
# Writes the given message to standard output
#
# $* - The message to write
#
msg() {
    echo "$*"
}

#
# Writes the given information message to standard output
#
# $* - The message to write
#
info() {
    msg "INFO: $*"
}

#
# Writes the given error message to standard error
#
# $* - The message to write
#
error() {
    msg "ERROR: $*" 1>&2
}
```

```
#
# main
#

# log something to stdout
info "running failure_sample_arg_exit_code.sh"

argexitcode=$1

# log to stderr
error "script failed, returning exit code ${argexitcode}"

# exit with specified exit code
exit ${argexitcode}
```

詳細なエラーの例

次の例では 'フックの失敗をより詳細なロギングで処理する方法を示します

```
#!/bin/sh

# failure_sample_verbose.sh
#
# A simple failure hook script with args for testing purposes.
#
# args: [The number of lines to output to stdout]

#
# Writes the given message to standard output
#
# $* - The message to write
#
msg() {
    echo "$*"
}

#
# Writes the given information message to standard output
#
# $* - The message to write
#
```

```

info() {
    msg "INFO: $*"
}

#
# Writes the given error message to standard error
#
# $* - The message to write
#
error() {
    msg "ERROR: $*" 1>&2
}

#
# main
#

# log something to stdout
info "running failure_sample_verbose.sh"

# output arg value to stdout
linecount=$1
info "line count ${linecount}"

# write out a line to stdout based on line count arg
i=1
while [ "$i" -le ${linecount} ]; do
    info "This is line ${i} from failure_sample_verbose.sh"
    i=$(( i + 1 ))
done

error "exiting with error code 8"
exit 8

```

終了コード例を使用した失敗

次の例は、終了コードを使用したフックの失敗を示しています。

```

#!/bin/sh

# failure_sample_arg_exit_code.sh
#
# A simple failure hook script for testing purposes.

```

```

#
# args: [the exit code to return]
#

#
# Writes the given message to standard output
#
# $* - The message to write
#
msg() {
    echo "$*"
}

#
# Writes the given information message to standard output
#
# $* - The message to write
#
info() {
    msg "INFO: $*"
}

#
# Writes the given error message to standard error
#
# $* - The message to write
#
error() {
    msg "ERROR: $*" 1>&2
}

#
# main
#

# log something to stdout
info "running failure_sample_arg_exit_code.sh"

argexitcode=$1

# log to stderr
error "script failed, returning exit code ${argexitcode}"

# exit with specified exit code

```

失敗後の成功の例

次の例では、最初の実行時にフックが失敗していますが、2 回目の実行後に成功しています。

```
#!/bin/sh

# failure_then_success_sample.sh
#
# A hook script that fails on initial run but succeeds on second run for
# testing purposes.
#
# Helpful for testing retry logic for post hooks.
#
# args: None
#
#
# Writes the given message to standard output
#
# $* - The message to write
#
msg() {
    echo "$*"
}

#
# Writes the given information message to standard output
#
# $* - The message to write
#
info() {
    msg "INFO: $*"
}

#
# Writes the given error message to standard error
#
# $* - The message to write
#
error() {
    msg "ERROR: $*" 1>&2
}
```

```
#
# main
#

# log something to stdout
info "running failure_success sample.sh"


if [ -e /tmp/hook-test.junk ] ; then
    info "File does exist. Removing /tmp/hook-test.junk"
    rm /tmp/hook-test.junk
    info "Second run so returning exit code 0"
    exit 0
else
    info "File does not exist. Creating /tmp/hook-test.junk"
    echo "test" > /tmp/hook-test.junk
    error "Failed first run, returning exit code 5"
    exit 5
fi
```

著作権に関する情報

Copyright © 2023 NetApp, Inc. All Rights Reserved. Printed in the U.S. このドキュメントは著作権によって保護されています。著作権所有者の書面による事前承諾がある場合を除き、画像媒体、電子媒体、および写真複写、記録媒体、テープ媒体、電子検索システムへの組み込みを含む機械媒体など、いかなる形式および方法による複製も禁止します。

ネットアップの著作物から派生したソフトウェアは、次に示す使用許諾条項および免責条項の対象となります。

このソフトウェアは、ネットアップによって「現状のまま」提供されています。ネットアップは明示的な保証、または商品性および特定目的に対する適合性の暗示的保証を含み、かつこれに限定されないいかなる暗示的な保証も行いません。ネットアップは、代替品または代替サービスの調達、使用不能、データ損失、利益損失、業務中断を含み、かつこれに限定されない、このソフトウェアの使用により生じたすべての直接的損害、間接的損害、偶発的損害、特別損害、懲罰的損害、必然的損害の発生に対して、損失の発生の可能性が通知されていたとしても、その発生理由、根拠とする責任論、契約の有無、厳格責任、不法行為（過失またはそうでない場合を含む）にかかわらず、一切の責任を負いません。

ネットアップは、ここに記載されているすべての製品に対する変更を随時、予告なく行う権利を保有します。ネットアップによる明示的な書面による合意がある場合を除き、ここに記載されている製品の使用により生じる責任および義務に対して、ネットアップは責任を負いません。この製品の使用または購入は、ネットアップの特許権、商標権、または他の知的所有権に基づくライセンスの供与とはみなされません。

このマニュアルに記載されている製品は、1つ以上の米国特許、その他の国の特許、および出願中の特許によって保護されている場合があります。

権利の制限について：政府による使用、複製、開示は、DFARS 252.227-7013（2014年2月）およびFAR 5252.227-19（2007年12月）のRights in Technical Data -Noncommercial Items（技術データ - 非商用品目に関する諸権利）条項の(b)(3)項、に規定された制限が適用されます。

本書に含まれるデータは商用製品および / または商用サービス（FAR 2.101の定義に基づく）に関係し、データの所有権はNetApp, Inc.にあります。本契約に基づき提供されるすべてのネットアップの技術データおよびコンピュータ ソフトウェアは、商用目的であり、私費のみで開発されたものです。米国政府は本データに対し、非独占的かつ移転およびサブライセンス不可で、全世界を対象とする取り消し不能の制限付き使用权を有し、本データの提供の根拠となった米国政府契約に関連し、当該契約の裏付けとする場合にのみ本データを使用できます。前述の場合を除き、NetApp, Inc.の書面による許可を事前に得ることなく、本データを使用、開示、転載、改変するほか、上演または展示することはできません。国防総省にかかる米国政府のデータ使用权については、DFARS 252.227-7015(b)項（2014年2月）で定められた権利のみが認められます。

商標に関する情報

NetApp、NetAppのロゴ、<http://www.netapp.com/TM>に記載されているマークは、NetApp, Inc.の商標です。その他の会社名と製品名は、それを所有する各社の商標である場合があります。